Module 11
Linear Collections: Stacks and Queues

Tamer Abdou, PhD

IND 830 - Python Programming for Data Science

Data Science Laboratory

Ryerson University

Ryerson University
The Chang School of Continuing Education

Outline

1. Stacks
2. Queues

Ryerson University
The Chang School of Continuing Education

Stacks

1 Stacks
Overview of Stacks
Implementing Stacks
Application of Stacks

2 Queues

Ryerson University The Chang School of Continuing Education

Stacks

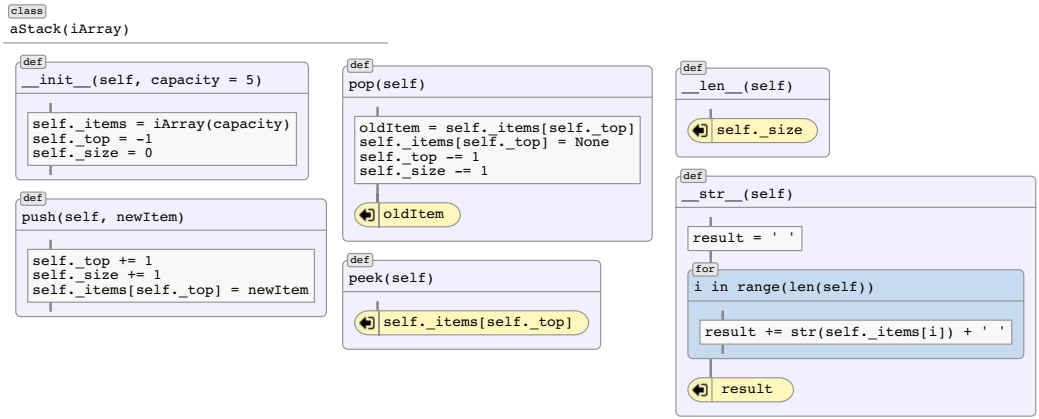
- Stacks are linear collections in which access is completely restricted to just one end, called the **top**
- Stacks adhere to a last-in first-out protocol (LIFO)
- Operations:
  push(stack, item): Inserts an item at the top of the stack

  pop(stack): Removes and returns the item at top, given the stack is not empty

  peek(stack): Returns the item at top, given the stack is not empty

Ryerson University The Chang School of Continuing Education

Stacks Interface



Ryerson University The Chang School of Continuing Education

Initializing an Empty Stack

```
class aStack(iArray):
    def __init__(self, capacity = 5):
        self._items = iArray(capacity)
        self._top = -1
        self._size = 0
    def push(self, newItem):
        self._top += 1
        self._size += 1
        self._items[self._top] = newItem
    def pop(self):
        oldItem = self._items[self._top]
        self._items[self._top] = None
        self._top -= 1
        self._size -= 1
        return oldItem
    def peek(self):
        return self._items[self._top]
    def __len__(self):
        return self._size
    def __str__(self):
        result = ' '
        for i in range(len(self)):
            result += str(self._items[i]) + ' '
        return result

s = aStack(4)
```

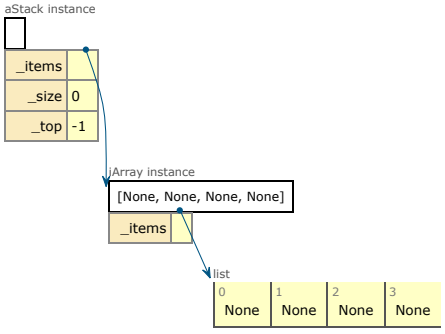


Figure 11.1: An empty stack with a load factor of 0

Ryerson University The Chang School of Continuing Education

Inserting an Element into a Stack

```
s = aStack(4)
'''
def push(self, newItem):
    self._top += 1
    self._size += 1
    self._items[self._top] = newItem
'''
iStr = '830'
for i in range(len(iStr)):
    s.push(iStr[i])
```

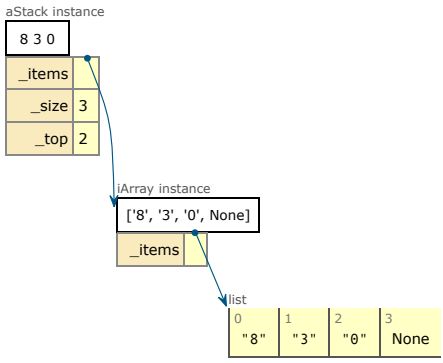


Figure 11.2: A stack with a load factor of 0.75

Ryerson University The Chang School of Continuing Education

Removing an Element from a Stack

```
s = aStack(4)
'''
def push(self, newItem):
    self._top += 1
    self._size += 1
    self._items[self._top] = newItem
'''
iStr = '830'
for i in range(len(iStr)):
    s.push(iStr[i])

'''
def pop(self):
    oldItem = self._items[self._top]
    self._items[self._top] = None
    self._top -= 1
    self._size -= 1
    return oldItem
'''
s.pop()
s.pop()
```

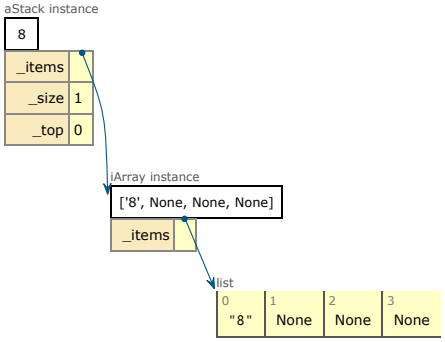


Figure 11.3: A stack with a load factor of 0.25

Ryerson University The Chang School of Continuing Education

Reaching a Full Stack

```
s = aStack(4)
'''
def push(self, newItem):
    self._top += 1
    self._size += 1
    self._items[self._top] = newItem
'''
iStr = '830'
for i in range(len(iStr)):
    s.push(iStr[i])

'''
def __len__(self):
    return self._size
 def peek(self):
    return self._items[self._top]
    def __len__(self):
'''
s.push('0')

print(s.peek())

print(len(s) == len(s._items))
```

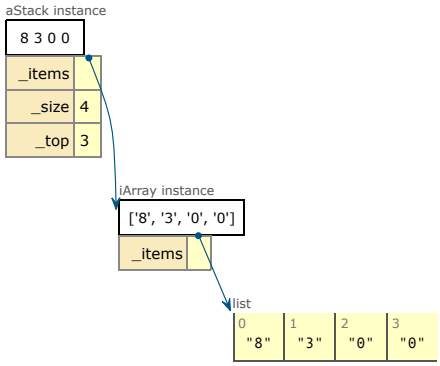


Figure 11.4: A full stack with a load factor of 1

Evaluating Arithmetic Expressions

```
Algorithm evaluatingPostfixForms():
    create an empty stack
    convert the infix form to its postfix equivalent form
    iterate through all the elements from left to right
          if the element is an operand
                push the element into the stack
          else:
             if the element is an operator
                   pop the top two operands from the stack
                   apply the operator to the two operands just popped
                   push the resulting value into the stack
    return the value at the top of the stack
```

Infix Evaluation: $8/4 + 5 * 3 \rightarrow 2 + 5 * 3 \rightarrow 2 + 15 \rightarrow 17$

Postfix Evaluation: 8 4 / 5 3 $*$ $+ \rightarrow$ 2 5 3 $*$ $+ \rightarrow$ 2 15 $+ \rightarrow$ 17

Ryerson University The Chang School of Continuing Education

Queues

1 Stacks

2 Queues
Overview of Queues
Implementing Queues
Application of Queues

Ryerson University The Chang School of Continuing Education

Queues

- Queues are linear collections in which access is completely restricted to just one end, called the **rear**
- Queues adhere to a first-in first-out protocol (FIFO)
- Operations:

  enqueue(Queue, item): Inserts an item to the rear of a queue

  dequeue(Queue): Removes and returns the item at the front

  peek(Queue): Returns the item at the front, given the queue is not empty

Ryerson University The Chang School of Continuing Education

Queues Interface

class
aQueue(iArray)

def
__init__(self, capacity = 5)

```
self._items = iArray(capacity)
self._rear = -1
self._size = 0
```

def
enqueue(self, newItem)

```
self._rear += 1
self._size += 1
self._items[self._rear] = newItem
```

def
dequeue(self)

```
oldItem = self._items[0]
```

for
i in range(len(self) - 1)

```
self._items[i] = self._items[i + 1]
self._items[i + 1] = None
```

```
self._rear -= 1
self._size -= 1
```

return oldItem

def
peek(self)

return self._items[0]

def
__len__(self)

return self._size

def
__str__(self)

```
result = ' '
```

for
i in range(len(self))

```
result += str(self._items[i]) + ' '
```

return result

Ryerson University The Chang School of Continuing Education

Initializing an Empty Queue

```
class aQueue(iArray):
    def __init__(self, capacity = 5):
        self._items = iArray(capacity)
        self._rear = -1
        self._size = 0
    def enqueue(self, newItem):
        self._rear += 1
        self._size += 1
        self._items[self._rear] = newItem
    def dequeue(self):
        oldItem = self._items[0]
        for i in range(len(self) - 1):
            self._items[i] = self._items[i + 1]
            self._items[i + 1] = None
        self._rear -= 1
        self._size -= 1
        return oldItem
    def peek(self):
        return self._items[0]
    def __len__(self):
        return self._size
    def __str__(self):
        result = ' '
        for i in range(len(self)):
            result += str(self._items[i]) + ' '
        return result

q = aQueue(4)
```

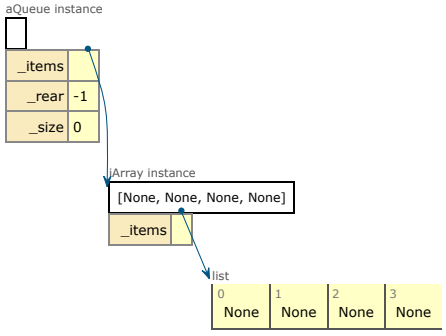


Figure 11.5: An empty queue with a load factor of 0

Ryerson University The Chang School of Continuing Education

Inserting an Element into a Queue

```
q = aQueue(4)
'''
def enqueue(self, newItem):
    self._rear += 1
    self._size += 1
    self._items[self._rear] = newItem
'''
iStr = '830'
for i in range(len(iStr)):
    q.enqueue(iStr[i])
```

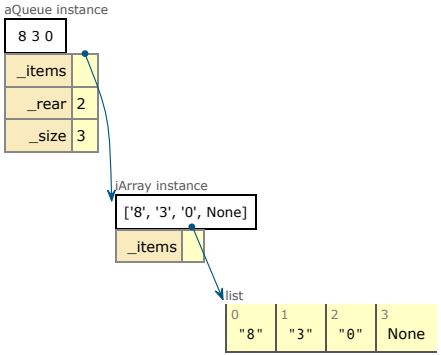


Figure 11.6: A queue with a load factor of 0.75

Ryerson University The Chang School of Continuing Education

Removing an Element from a Queue

```
q = aQueue(4)
'''
def enqueue(self, newItem):
    self._rear += 1
    self._size += 1
    self._items[self._rear] = newItem
'''
iStr = '830'
for i in range(len(iStr)):
    q.enqueue(iStr[i])

'''
def dequeue(self):
    oldItem = self._items[0]
    for i in range(len(self) - 1):
        self._items[i] = self._items[i + 1]
    self._rear -= 1
    self._size -= 1
    return oldItem
'''
q.dequeue()
q.dequeue()
```

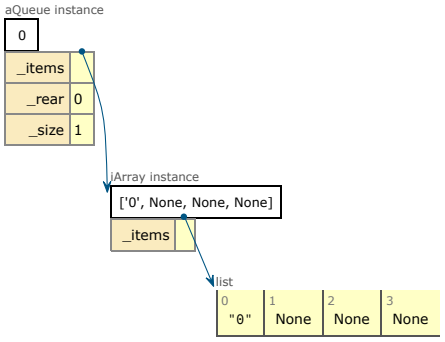


Figure 11.7: A queue with a load factor of 0.25

Ryerson University The Chang School of Continuing Education

Reaching a Full Queue

```
q = aQueue(4)
'''
def enqueue(self, newItem):
    self._rear += 1
    self._size += 1
    self._items[self._rear] = newItem
'''
iStr = '830'
for i in range(len(iStr)):
    q.enqueue(iStr[i])

'''
def __len__(self):
    return self._size
def peek(self):
    return self._items[0]
'''
q.enqueue('0')

print(q.peek())

print(len(q) == len(q._items))
```

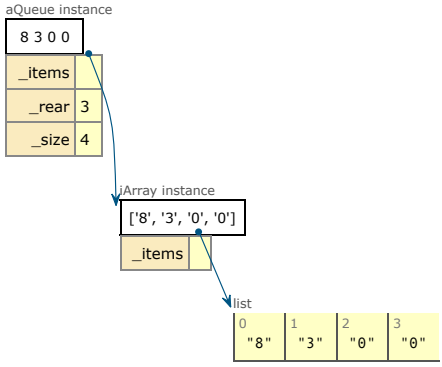


Figure 11.8: A full queue with a load factor of 1

Priority Queues

- A priority queue is a special type of queue structure
- Similar to queues, priority queues adhere to a first-in first-out protocol (FIFO)
- Unlike queues:
  - Priority queues rank elements before inserting them into the queue
  - Elements of higher priority are removed before those of lower priority
  - Elements of equal priority are removed in the usual FIFO order

Ryerson University The Chang School of Continuing Education

CPU Scheduling

Table 11.1: Duration of processes with a time quantum of 2 units

Process ID	Arrival Time	Required Time
P_1	0	4
P_2	1	3
P_3	2	2
P_4	3	1

Table 11.2: CPU scheduler during run-time

Queue	Unit Time
Empty	0
P_2	1
P_3P_1	2
$P_3P_1P_4$	3
$P_1P_4P_2$	4
$P_1P_4P_2$	5
P_4P_2	6
P_4P_2	7
P_2	8
Empty	9
Empty	10

Ryerson University The Chang School of Continuing Education

References

- Lambert, K. A. (2010). Linear collections: Stacks. In *Fundamentals of Python: From first programs through data structures* (pp. 561-583). Boston, MA: Cengage Learning
- Lambert, K. A. (2010). Linear collections: Queues. In *Fundamentals of Python: From first programs through data structures* (pp. 603-629). Boston, MA: Cengage Learning
- Python Software Foundation. (2020). Python 3.7.8 documentation. Retrieved from `https://docs.python.org/3.7/index.html`.